Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

Treasury Oversight Mechanism

Audit Report

MLabs Audit Team

July 15, 2025

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

Contents

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

1 Disclaimer

This audit report is presented without warranty or guarantee of any type. Neither MLabs nor its auditors can assume any liability whatsoever for the use, deployment or operations of the audited code. This report lists the most salient concerns that have become apparent to MLabs' auditors after an inspection of the project's codebase and documentation, given the time available for the audit. Corrections may arise, including the revision of incorrectly reported issues. Therefore, MLabs advises against making any business or other decisions based on the contents of this report.

An audit does not guarantee security. Reasoning about security requires careful considerations about the capabilities of the assumed adversaries. These assumptions and the time bounds of the audit can impose realistic constraints on the exhaustiveness of the audit process. Furthermore, the audit process involves, amongst others, manual inspection and work which is subject to human error.

MLabs does not recommend for or against the use of any work or supplier mentioned in this report. This report focuses on the technical implementation provided by the project's contractors and subcontractors, based on the information they provided, and is not meant to assess the concept, mathematical validity, or business validity of their product. This report does not assess the implementation regarding financial viability nor suitability for any purpose. *MLabs does not accept responsibility for any loss or damage howsoever arising which may be suffered as result of using the report nor does it guarantee any particular outcome in respect of using the code on the smart contract.*

2 Background

2.1 Scope

During the audit, MLabs has inspected the code contained in the provided files and attempted to locate problems that can be found in the following categories:

- Unclear or wrong specifications which could lead to unwanted behaviour.
- Wrongful implementation.
- Vulnerabilities that can be leveraged by an attacker.

Where possible, MLabs has provided recommendations to address the relevant issues.

2.2 Parties Involved

In this document, we refer to the MLabs Audit Team as MLabs.

2.3 Methodology

2.3.1 Information

MLabs analysed the files shared by the SundaeSwap team containing the implementation of the protocol. The protocol is implemented in treasury-contracts. The on-chain scripts that have been audited can be found in the `lib` and `validators` directories of the repo. The protocol also relies on aicone to authorize multi signatures on-chain. Reviewing the aicone code was not in scope for this audit. All the code is written in Aiken. Review of the Aiken language is not considered in scope of this audit.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

2.3.2 Audited File Checksums

The following checksums are those of the shared files from the `treasury-contracts` These were generated using the following sha256 binary:

```
$ sha256sum --version

sha256sum (GNU coreutils) 9.1
Packaged by https://nixos.org
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Ulrich Drepper, Scott Miller, and David Madore.
```

The checksums of the reviewed files are:

```
bf06f3d...7b28faa  lib/logic/treasury/withdraw.ak
4df8932...884a56c  lib/logic/treasury/fund.ak
d6b831e...be1b316  lib/logic/treasury/disburse.ak
b6ce04b...2b8a217  lib/logic/treasury/reorganize.ak
641c052...af4dcda  lib/logic/treasury/sweep.ak
a94d8d5...40a3985  lib/logic/vendor/withdraw.ak
d199a6b...754f256  lib/logic/vendor/malformed.ak
3ee1cdb...9a3dbab  lib/logic/vendor/adjudicate.ak
aa5cf3f...4faa445  lib/logic/vendor/modify.ak
c623f85...68db041  lib/logic/vendor/sweep.ak
24a9f11...4964ace  lib/utilities.ak
f3e10ce...eccfe41  validators/vendor.ak
acc76c5...e33e589  validators/oneshot.ak
99976ea...f8c6b60  validators/treasury.ak
```

2.3.3 Audit Timeline

The audit process undertaken by MLabs was divided into three distinct phases. The timeline was structured as follows:

Study of the protocol and it's requirements (Weeks 1): the focus of this phase was gathering information on the protocol. Documentation and the provided repository were studied to ensure a full understanding of the protocols intended behavior. After this, the MLabs team came up with a list of properties that the protocol should enforce and carried out an assessment of the current test coverage.

In-depth review of the protocol (Weeks 2 to 3): in this phase the MLabs team focussed on identifying possible vulnerabilities with the protocol. The protocol was split into different parts, and team members took a round robin approach at reviewing each part in depth, raising possible vulnerabilities identified in issues. During this phase some tests were also written to verify the vulnerabilities found.

Test finalisation and writing the report (Weeks 4): the focus of this phase was to write tests to prove or disprove the potential vulnerabilities found in the previous phase. Finally, the last part of the phase consisted in formalising the audit findings into a detailed report.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

2.4 Metrics

2.4.1 CVSS

The audit used the Common Vulnerability Scoring System and the NVD Calculator to provide a standardised measure for the severity of the identified vulnerabilities. Although MLabs recognises that some of the parameters of the tools may not be relevant for the audit of a Cardano protocol, the team believes that using a standard is still valuable in providing a more unbiased severity metric for the findings.

2.4.2 Severity Levels

The aforementioned CVSS calculations were then benchmarked using the CVSS-Scale metric, receiving a grade spanning from `Low` to `Critical`. This additional metric allows for an easier, human understandable grading, whilst leveraging the CVSS standardised format.

3 Audit

3.1 Executive Summary

The audit findings can be categorised as the following types:

1. deviation-from-spec ×5
2. unbounded-datum ×5
3. multiple-satisfaction ×2
4. insufficient-staking-control ×1
5. min-ada ×1
6. utxo-contention ×1

3.2 Important assumptions

This audit relies on the following assumptions:

- There are no bugs and issues with `Aiken` that can cause vulnerabilities
- `aicone/multisig` behaves as intended and does not present any vulnerability

3.2.1 Script Parameters

As mentioned in previously, the protocol relies on `aicone/multisig` to encode different authorisation schemes on-chain. Specifically, the `TreasuryConfiguration` and `VendorConfiguration`, which respectively parametrise the treasury and vendor scripts, both have a `permissions` field encoding the authorisation required for the different redeemers (`sweep`, `fund`, `modify`, etc). The datum in a vendor UTxO, also contains a `vendor` field which uses the same representation to store the authorisation required for the vendor that will be checked in `withdraw`. Authorisations are encoded through a `MultisigScript` from `aicone/multisig` which has the following definition:

```
pub type MultisigScript {
  Signature { key_hash: ByteArray }
  AllOf { scripts: List<MultisigScript> }
  AnyOf { scripts: List<MultisigScript> }
```

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

```
  AtLeast { required: Int, scripts: List<MultisigScript> }
  Before { time: Int }
  After { time: Int }
  Script { script_hash: ByteArray }
}
```

This DSL essentially replicates what is expressible through Cardano Native Scripts, but also adds the `Script` constructor: which checks if there is a withdrawal happening at a given script address. This essentially gives the `MultisigScript` power to yield to an arbitrary script for authorisation.

All this means that the security of the protocol, ultimately relies on how these `MultisigScripts` are created at contract instantiation time. Furthermore, if this `MultisigScript` yields to a different script, then it is important to keep in mind this different script has not necessarily been audited.

MLabs has agreed to review the parameters that will be used to instantiate the contracts (see On-chain parameter review). We also recommend the parameters are made available to users of the protocol, so that they can always do this verification on their own. This is necessary as each instance of treasury and vendor contracts that are "deployed", need to be verified independently.

3.3 Vulnerabilities

The following section provides details on each of the vulnerabilities found during the audit.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

3.4 Double satisfaction between 2 TRSC instances when sweeping both

Severity	CVSS	Vulnerability type
Critical	*9.1*	*multiple-satisfaction*

3.4.1 Property

Anyone can sweep the remaining ADA from the treasury reserve account back to the Cardano treasury after the reserve account's expiration date.

3.4.2 Problem

When sweeping 2 TRSC instances, the treasury donation can be satisfied by the first one, and then the same amount of funds can be stolen from the second one.

3.4.3 Reason for the problem

The validator is summing up only UTXOs from a single TRSC instance. This causes the issue that there can be multiple TRSC instances that allow spending UTXOs.

The vulnerability is made more severe by the fact that, after TRSC expiration, anyone can sweep TRSC.

3.4.4 How to reproduce

Reproduced in this test

3.4.5 Suggestion to fix the problem

Disallow spending other script inputs than the ones from the current TRSC instance.

3.4.6 Implemented solution

Fixed according to suggestion.

Both treasury and vendor scripts now enforce that all script inputs come from the credentials specified in the registry.

3.5 Steal funds when sweeping tresury

Severity	CVSS	Vulnerability type
High	*7.9*	*multiple-satisfaction*

3.5.1 Property

Anyone can sweep the remaining ADA from the treasury reserve account back to the Cardano treasury after the reserve account's expiration date.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

3.5.2 Problem

The sweep validator for the treasury always assumes that the sum of all inputs from the treasury is going to be greater than the sum of all outputs. This is not always the case when combined with other operations, and can lead to someone stealing funds.

3.5.3 Reason for the problem

The sweep redeemer enforces that, under some conditions, funds can be swept back from the treasury contract into the Cardano treasury with a donation. This donation has to be greater than the sum of all treasury inputs minus the sum of all treasury outputs. When combining this with another operation that produces outputs at the treasury, we can make the difference in inputs and outputs arbitrarily small, or even negative. For example, suppose we are claiming a malformed utxo at the vendor address containing 10 Ada. This Ada is supposed to go back to the treasury contract, which the malformed redeemer correctly ensures. If, at the same time, we are sweeping a utxo with 5 Ada from the treasury contract, we will end up with a single input coming from the treasury (the utxo being swept) and a single output going to the treasury (the malformed vendor utxo). In this case, the `sweep` redeemer will check that the donation is >= 5 - 10, which is true for any arbitrarily small donation (which has to be positive)

3.5.4 How to reproduce

Reproduced in this test

3.5.5 Suggestion to fix the problem

Treasury sweep should ensure that the only script UTXOs spent are from the current treasury. This would prevent double satisfaction when interacting with multiple scripts.

3.5.6 Implemented solution

Fixed according to suggestion.

Both `sweep` and `malformed` now enforce there are no inputs from other scripts.

3.6 Steal withdraw rewards from treasury contract with double satisfaction

Severity	CVSS	Vulnerability type
High	*7.9*	*multiple-satisfaction*

3.6.1 Property

Any staking rewards accrued using the TRSC staking credentials must be returned to the TRSC

3.6.2 Problem

It is possible to use another transaction that sends lovelace to the treasury to appropriate funds from the treasury contract rewards account

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

3.6.3 Reason for the problem

The withdrawal code tries to prevent a double satisfaction by asserting there are no inputs from the treasury, this works only partially. If an attacker were to combine this with trying to collect a malformed vendor utxo back to the treasury, then the lovelace flowing in from the malformed vendor UTxO can satisfy the withdraw check (which is: lovelace going to treasury is >= than the withdrawal tx entry).

3.6.4 How to reproduce

Reproduced in this test

3.6.5 Suggestion to fix the problem

Ensure there are no script inputs spent as part of this TX.

3.6.6 Implemented solution

Fixed according to suggestion.

Both `withdraw` and `malformed` now enforce there are no inputs from other scripts.

3.7 Uncapped `VendorDatum` size can halt vendor script

Severity	CVSS	Vulnerability type
High	*7.0*	*unbounded-datum*

3.7.1 Property

3.7.2 Problem

Funds could get locked in the vendor script because any vendor script action cannot fit on-chain limits due to the vendor datum's size.

3.7.3 Reason for the problem

The `VendorDatum` contains the field `payouts` of type `List<...>`, where the list can grow up to the size that the fund action allows. This means that during other actions in the vendor contract, the action can be blocked because the action exceeds the on-chain limits due to the list size. This way the funds could stay locked in that UTXO without the possibility to either withdraw, sweep, modify or adjudicate. Because of this, we suggest introducing limits on the length of the list.

3.7.4 How to reproduce

Tested here

Run with:

```
$ bun test --test-name-pattern "With uncapped datum"
```

The vendor's `Withdraw` and `Sweep` endpoints are affected. `Adjudicate` does NOT seem to be affected (it is possible to adjudicate from one to all payouts).

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

3.7.5 Suggestion to fix the problem

Set multiple caps on the `payouts` field in the `VendorDatum` type based on the benchmarks. Both of these caps should be set in both the fund and modify actions when the datum can be modified. Do not forget that not only do `payouts` need to be capped, but also the nested field `value` in the `Payout` type.

3.7.6 Implemented solution

Fixed according to suggestion.

Checks are now in place to prevent the datum from growing unbounded. Payouts are capped to contain up to 4 different assets (these can be different within payouts for the same contract), and the number of payouts is capped at 24.

3.8 Modify active matured payouts in datum is possible

Severity	CVSS	Vulnerability type
High	*7.5*	*deviation-from-spec*

3.8.1 Property

The committee can modify the vendor contract in ways that include: modifying the payout structure, or cancelling the contract. This should not prevent the vendor from being able to claim already matured payouts.

3.8.2 Problem

Modifying active matured payouts is possible, and when doing so, a matured payout's maturation date can be changed arbitrarily, moving it from the matured state to unmatured. This changes the state of the payout from claimable to non-claimable, which shouldn't be allowed by the property.

3.8.3 Reason for the problem

The source of the problem is that there's no check for the matured payouts on the vendor's output datum, and those can be set arbitrarily.

3.8.4 How to reproduce

Reproduced in this test case. It demonstrates that the second matured payout can become unmatured during `modify` action.

3.8.5 Suggestion to fix the problem

Enforce that the active matured payouts in the datum stay unchanged after any modify action. The only payouts that can be modified are the unmatured or paused ones.

3.8.6 Implemented solution

Not fixed.

The reasoning is that, since the vendor is signing this transaction, they are agreeing to any modification occurring to the datum. In particular, they can choose to forgo a payment they are owed and have it returned to the treasury.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

3.9 Modification forces withdrawal of matured non-ada payouts

Severity	CVSS	Vulnerability type
High	*7.5*	*deviation-from-spec*

3.9.1 Property

The committee can modify the vendor contract in ways that include: changing the payout structure, or cancelling the contract. This should not prevent the vendor from being able to claim already matured payouts

3.9.2 Problem

Non-ADA matured active unwithdrawn payouts have to be withdrawn, while ones denominated in ADA can be kept in the vendor UTxO during modify.

3.9.3 Reason for the problem

The problem lies in the asymmetry of `equal_plus_min_ada`, which forces the sum of non-ADA outputs at the vendor and treasury outputs to be exactly equal to the unmatured value, while allowing ADA to be bigger. This means that non-ADA payouts can not be returned to the vendor UTxO to be withdrawn later, but must be withdrawn while modifying.

3.9.4 How to reproduce

Reproduced in this case. This test case fails, demonstrating that matured funds denominated in non-ADA assets cannot be locked back to the vendor UTXO during modify to be later still claimable by the vendor.

3.9.5 Suggestion to fix the problem

The following check,

```
equal_plus_min_ada(
        unmatured_value,
        assets.merge(vendor_output_sum, treasury_output_sum),
      ),
```

should treat ADA and non-ADA assets equally. It should enforce that there are at least non-ADA assets in `assets.merge(vendor_output_sum, treasury_output_sum)` to cover for `unmatured_value`, but also allow for more.

3.9.6 Implemented solution

Fixed according to suggestion.

3.10 Vendor datum can desynchronize with value during modify

Severity	CVSS	Vulnerability type
High	*7.5*	*deviation-from-spec*

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

3.10.1 Property

A vendor contract must always contain enough assets to cover all the payouts

3.10.2 Problem

Payouts in datum can have a different total value than the value available in the UTXO. This could halt the protocol, since the withdraw action requires *all* value of the mature payouts to be withdrawn.

3.10.3 Reason for the problem

The funds can move to a new UTXO, and there's no constraint on the datum. This could cause locking the funds forever (in case funds are matured, but the payout dictates a higher value than available in the UTXO, the withdrawal then can't succeed, and the funds stay locked).

3.10.4 How to reproduce

Reproduced in this test.

3.10.5 Suggestion to fix the problem

The vendor's output datum has to be properly constrained. Right now, there's no constraint whatsoever.

3.10.6 Implemented solution

Fixed according to suggestion.

There is now a check to ensure that the vendor output has at least enough funds to cover the new payouts after `modify`.

3.11 Datum is not checked during fund action

Severity	CVSS	Vulnerability type
High	*7.5*	*unbounded-datum*

3.11.1 Property

A malformed datum on the vendor utxo can lead to several violations of the protocol properties.

3.11.2 Problem

There are no checks on the datum of the utxo being locked in the vendor script. This can cause several issues that stem from a malformed vendor utxo, such as:

- Vendor datum's vendor permissions can be set to an arbitrary value, i.e., the vendor can be unable to withdraw eligible payouts.
- Payout maturation dates are not constrained enough, and can be set after the expiration of the vendor script. These misconfigured payouts can be swept even if they are active and mature (described in detail in - *Committee can sweep mature payouts after expiry if they are misconfigured*).
- The funds of payouts can be denominated in any token (described in detail as a separate issue - *Fund should check redeemer is only denominated in the accepted currencies*)

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

- A very big number of payouts can be set, halting some endpoints down the line due to script execution consuming too much memory (described in more detail here in - *Uncapped `VendorDatum` size can halt vendor script*)

3.11.3 Reason for the problem

There are not enough checks on the VendorDatum in the fund logic.

3.11.4 How to reproduce

Payout maturation dates after vendor script expiration are reproduced in this test.

3.11.5 Suggestion to fix the problem

The VendorDatum should not only be expected to be of the correct type, but some invariants should be verified.

- The payout maturation dates should not only be before the treasury expiration, but should be checked against the vendor datum as well
- The payouts should be checked only to be denominated in the accepted currencies (as described in the suggestion for the issue - *Fund should check redeemer is only denominated in the accepted currencies*)

It is also arguable that the vendor permissions being set should be checked. This could be done by requiring the vendor to witness the fund action by checking the permissions set in the VendorDatum.

3.11.6 Implemented solution

Fixed according to suggestion.

Except for limiting accepted currencies, which is deemed out of scope (see - *Fund should check redeemer is only denominated in the accepted currencies*)

3.12 Modify endpoint does not enforce invariants on the vendor datum

Severity	CVSS	Vulnerability type
High	*7.5*	*unbounded-datum*

3.12.1 Property

A malformed datum on the vendor utxo can lead to several violations of the protocol properties, as discussed below.

3.12.2 Problem

Modify is not constrained enough, allowing many changes to vendor UTxOs that could potentially break the protocol.

3.12.3 Reason for the problem

Some invariants are assumed to hold about the vendor UTxO, namely:

- It contains enough value to cover the payouts (described in detail in - *Vendor datum can desynchronize with value during modify*)

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

- Payout maturation dates are not constrained enough, and can be set after the expiration of the vendor script. These misconfigured payouts can be swept even if they are active and mature (described in detail in - *Committee can sweep mature payouts after expiry if they are misconfigured*).
- Its permissions are not changed.
- Its payouts should be denominated in valid tokens (ADA, USDA, USDM).

These invariants are only partially enforced on utxo creation (see issue *Datum is not checked during fund action*) and completely ignored in the case of UTxO modification.

Even assuming good faith, it is possible to make a mistake when signing transactions, and breaking some of the invariants above could lead to draining funds from the treasury and other vulnerabilities.

3.12.4 How to reproduce

This test shows that a payout date can be set after the expiry of the vendor contract

Tests for other invariants being broken are in their respective issues (*Vendor datum can desynchronize with value during modify* and *Committee can sweep mature payouts after expiry if they are misconfigured*).

3.12.5 Suggestion to fix the problem

Enforce all the invariants whenever a vendor UTxO is being created or modified.

3.12.6 Implemented solution

Fixed according to suggestion.

Except for limiting accepted currencies, which is deemed out of scope (see - *Fund should check redeemer is only denominated in the accepted currencies*).

3.13 Committee can pause matured payouts during adjudicate

Severity	CVSS	Vulnerability type
Medium	*6.5*	*deviation-from-spec*

3.13.1 Property

The vendor can always claim a payout that is active, provided the maturation date has passed. Withdrawal of multiple such payouts at once is allowed.

3.13.2 Problem

The committee can pause a payout, even if the maturation date has passed.

3.13.3 Reason for the problem

The problem is caused by the fact that, in the logic for `adjudicate`, we only consider a payment matured if the validity range of the tx is entirely after the payout maturation date, and the status is active. But the committee has full control over the lower bound of the validity range and can set it arbitrarily early, allowing them to ignore this check.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

3.13.4 How to reproduce

Reproduced in this test

3.13.5 Suggestion to fix the problem

In combination with the `is_entirely_after`, check the length of the validity range and cap it. That would ensure that the validity range could be extended to the history up to the point `now - interval_max_length`.

That would still allow some degree of traveling in time, but it would be allowed only to the degree that the max interval cap allows.

3.13.6 Implemented solution

Fixed according to suggestion.

The maximum allowed interval is now 36 hours; this still allows committee members to pause a payout that had matured in this time window. Ideally, the window could be made even smaller; however, there is some time required to gather all the required signatures off-chain, and imposing a shorter time window could lead to the committee being unable to gather all the signatures in time.

3.14 Committee can steal malformed vendor inputs while re-organizing treasury inputs

Severity	CVSS	Vulnerability type
Medium	*6.4*	*multiple-satisfaction*

3.14.1 Property

The committee can reorganize (split, merge) the treasury reserve account UTXOs only before the account expiration date

3.14.2 Problem

The committee can steal funds by combining reorganising the treasury input with claiming a malformed vendor utxo.

3.14.3 Reason for the problem

The reorganize logic checks that the sum of all inputs from the treasury is smaller than or equal to the sum of all outputs going to the treasury. However, when combining this with other operations that produce outputs at the treasury (such as claiming a malformed vendor input), a double satisfaction can happen, and the committee can steal the extra funds

3.14.4 How to reproduce

Reproduced in this test

3.14.5 Suggestion to fix the problem

Disallow other script inputs than the ones from the current script using the same redeemer.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

3.14.6 Implemented solution

Fixed according to suggestion.

Both `modify` and `reorganize` now enforce that there are no inputs from other scripts.

3.15 Staking credential can be attached to unswept value in vendor contract

Severity	CVSS	Vulnerability type
Medium	*4.3*	*insufficient-staking-control*

3.15.1 Property

Outgoing UTXOs to the vendor contract controlled by the validators' logic cannot have any staking credentials attached

3.15.2 Problem

During vendor sweep, unmatured value remaining in the vendor contract can get staking credential attached.

3.15.3 Reason for the problem

There's no check that the vendor output cannot have a staking credential.

3.15.4 How to reproduce

Reproduced in this test

3.15.5 Suggestion to fix the problem

Introduce a check to ensure the vendor's output cannot have a staking credential attached.

3.15.6 Implemented solution

Fixed, but differently than suggested.

Another reviewer pointed out that the Cardano constitution mandates that funds received from a Cardano treasury withdrawal must be delegated to the always abstain DRep. This includes funds in both the treasury and vendor contracts. The fix that has been applied is to enforce that funds locked in each script are always delegated to the script staking credential. Both scripts now allow registration/unregistration of the credential and (only) delegation to the AlwaysAbstain DRep.

3.16 minAda is sweepable from vendor utxo with unmatured payments

Severity	CVSS	Vulnerability type
Medium	*4.3*	*min-ada*

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

3.16.1 Property

Anyone can sweep remaining ADA from the treasury reserve account back to the Cardano treasury after the reserve account expiration date with no other cost than transaction fees

3.16.2 Problem

`minAda` is not correctly accounted for in the sweep vendor logic, forcing it to be all swept into the treasury even in the case of a partial sweep.

3.16.3 Reason for the problem

Suppose a vendor UTxO consists of only non-ADA payouts in the datum. Then the UTxO that holds those payouts is going to have a value of minAda + whatever is required to cover the payouts. If such UTxO expires with some matured and unmatured payouts, it is expected that any user can sweep back the unmatured payouts into the treasury contract, while leaving the matured payments in the UTxO locked at the vendor contract.

Given this definition of `swept_value`, i.e., the value that is being swept from the vendor utxo into the treasury, we also account for any minAda present in the `input.value` as part of what should be swept.

But in case of a partial-sweep, we expect there to be a continuing output at the vendor contract, which must also have minAda. In this case, the sweeper must cover the minAda for the continuing output.

3.16.4 How to reproduce

This shows an example scenario. Note that the front-end automatically includes the minAda value in the output, so there is no error in the transaction

3.16.5 Suggestion to fix the problem

We could use an approach similar to `withdraw` to partition the list into the matured and unmatured payouts. Then we could use this to calculate `swept_value` as

```
let swept_value = unmatured_payouts
      |> list.foldl(
          assets.zero,
          fn(payout, sum) {
            assets.merge(sum, assets.from_asset_list(payout.value))
          },
        )
```

3.16.6 Implemented solution

Fixed, but differently than suggested.

Treasury inputs are now completely disallowed when sweeping a vendor utxo. Note this still means that the sweeper will have to supply minAda to create the utxo at the treasury if they are sweeping only non-ADA assets.

3.17 DoSing treasury sweep UTxOs

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

Severity	CVSS	Vulnerability type
Medium	*4.3*	*utxo-contention*

3.17.1 Property

Anyone can sweep the remaining ADA from the treasury reserve account back to the Cardano treasury after the reserve account's expiration date.

3.17.2 Problem

Treasury sweep could be DoSed, and the funds can be swept 1 lovelace at a time.

3.17.3 Reason for the problem

There is no minimum enforced on the amount being donated to the Cardano treasury during sweep, which allows users to pick arbitrarily small amounts of lovelace to sweep, potentially DoSing the treasury contract UTxOs.

3.17.4 How to reproduce

The issue is reproduced in this test

3.17.5 Suggestion to fix the problem

Disallow donations that are "too small" relative to the input being swept. This is slightly complicated by the fact that we must account for non-ADA assets in treasury contract UTxOs: these assets can't be donated but must be returned to the same address, and as such, will need to contain a certain amount of ADA for the UTxO. In this case, we could either let the sweeper cover the cost of minAda (as the TODO in the comments mentions), or allow for some small constant amount of ADA to be retained in the input.

3.17.6 Implemented solution

Fixed according to suggestion.

The difference between the lovelace in the input, and the donation must be <= 5 ADA. This forces sweepers to donate almost all of the lovelace in the treasury input, but still allows flexibility for minAda increasing.

3.18 Fund should check redeemer is only denominated in the accepted currencies

Severity	CVSS	Vulnerability type
Low	*3.3*	*deviation-from-spec*

3.18.1 Property

Funds for a vendor project can be denominated only in ADA, USDa, or USDm

3.18.2 Problem

The allowed tokens are not constrained in the Fund redeemer, i.e., any token can be used as a payout. This is unconstrained both in the fund and modify logic.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

3.18.3 Reason for the problem

There is no check that the payouts stored in the vendor UTxO datums are all denominated in the accepted currencies. This applies both fund and modify.

3.18.4 How to reproduce

Reproduced in this test.

3.18.5 Suggestion to fix the problem

Allow only the allowed tokens to be in the payouts.

3.18.6 Implemented solution

Not fixed.

The reasoning is that the contracts are implemented generically to allow for different uses. It will be up to the committee to enforce only allowed assets in the payouts.

4 Recommendations for Code Improvement

In the following section we present targeted recommendations aimed at improving the code quality for on-chain code, rectifying discrepancies between the actual implementation and the initial documentation provided, and answering some of the questions left as comments in the code. While these issues do not directly result in exploitable vulnerabilities, they significantly impact the readability and maintainability of the code. Some deficiencies, if not addressed, could potentially lead to security vulnerabilities in the future. This section underscores the importance of proactive improvements to safeguard against potential exploits.

4.1 `payout_upperbound` from `TreasuryConfiguration` is unused

4.1.1 Problem

Like the title suggests, there is an unused variable named `payout_upperbound` in the `TreasuryConfiguration` type.

4.1.2 Reason for the problem

There shouldn't be any unused variables.

4.1.3 Suggestions

Either utilise the `payout_upperbound` or remove it completely.

4.1.4 Implemented solution

Fixed according to suggestion.

`fund` now enforces that the payouts are before the `payout_upperbound`; however, `modify` uses the `vendor.expiration` as an upper bound. This introduces a difference in how far in the future payouts can be configured between funding and modifying.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

4.2 Fund action isn't enforced to be witnessed by the vendor

4.2.1 Problem

The requirement is that the vendor has to witness the fund action, i.e., sign it along with the committee. However, the implementation isn't clear that it enforces the signature of the vendor since there's a single set of permissions in `TreasuryPermissions.fund`. Initialisation of this permission could miss the vendor.

4.2.2 Reason for the problem

Payout can be mistakenly assigned to a vendor since it's not enforced. Adding such a constraint on-chain will enforce that there's no mistake and additionally will help ensure the correct construction of the corresponding datum's field.

4.2.3 Suggestions

Make it explicit who the vendor is and that he must sign the transaction.

4.2.4 Implemented solution

Fixed according to suggestion.

The correct vendor permissions are now correctly checked when creating or modifying a vendor UTxO.

4.3 Modify doesn't allow for an increase of payouts

4.3.1 Problem

The payouts can't be modified by increasing the total value of payouts from the treasury.

4.3.2 Reason for the problem

This isn't allowed since inputs from the corresponding treasury are explicitly disallowed here.

Also, they can't be increased using a payout from other vendors in the same vendor contract instance, since during this TX, only a single input can be spent from the current vendor contract instance based on this check.

And when it would be fund from other places, it wouldn't pass the on-chain logic because of this check.

4.3.3 Suggestions

We think this functionality should be allowed. In that case, the logic has to be modified appropriately to work with that. Right now, there's no treasury input allowed, but it will be needed in case we want to allow an increase in payouts.

4.3.4 Implemented solution

Not fixed.

But a similar result can always be achieved by returning all the funds to the treasury in a first `modify` transaction, and then creating a new vendor contract with `fund`.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

4.4 Unnecessary redeemer parameter for adjudicate, fund and disburse

4.4.1 Problem

The `statuses` redeemer parameter seems to be redundant; such information is already in the output UTXOs, and this parameter doesn't seem to bring extra value.

The same applies to the `amount` parameter for the fund and disburse redeemers.

4.4.2 Reason for the problem

These redeemers are all redundant, as they are always already encoded in the transaction that is submitted.

4.4.3 Suggestions

Remove the redundant parameters.

4.4.4 Implemented solution

Not fixed.

4.5 Disallow spending treasury UTxOs during vendor sweep

4.5.1 Problem

It is not necessary, as treasury UTxOs can always be reorganized, but more importantly, there is no corresponding redeemer in the treasury, and it is unclear what should be used here.

4.5.2 Reason for the problem

This unnecessarily increases complexity and could introduce issues.

4.5.3 Suggestions

Disallow spending treasury inputs during vendor sweep.

4.5.4 Implemented solution

Fixed according to suggestion.

No treasury input is allowed during sweep.

4.6 Redundant checks in treasury sweep

4.6.1 Problem

Redundant checks in sweep unnecessarily increase complexity and also enable polluting the treasury UTxO with dummy tokens.

4.6.2 Reason for the problem

This constraint seems to be unnecessarily complex.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

4.6.3 Suggestions

It could be replaced by the following:

```
expect without_lovelace(input_sum) == without_lovelace(output_sum)
```

4.6.4 Implemented solution

Not fixed.

4.7 Limit oneshot minting policy to mint a single registry token

4.7.1 Problem

The oneshot minting policy, which is used to mint the registry token NFT, does not limit the mint to a single token. This means that potentially there might be multiple tokens with the same `PolicyId`, but different datums, which can lead to some issues.

4.7.2 Reason for the problem

Since the minting policy of the script hash registry allows for minting multiple tokens, this makes it possible to have two different registries holding the same token. This would most likely be accidental, but such a mistake can lead to potential issues.

4.7.3 Suggestion to fix the problem

Fix the minting policy used for the script hash registry and enforce only a single token is being minted.

4.7.4 Implemented solution

Fixed according to suggestion.

4.8 `equal_plus_min_ada` ADA amount uncapped

4.8.1 Problem

`equal_plus_min_ada` allows any ADA amount higher than the expected one.

4.8.2 Reason for the problem

In case of a malicious frontend/off-chain, ADA could be stolen from the wallet of the TX submitter. There could be a constant representing min ADA, e.g., 5 ADA, that could comfortably satisfy min ADA for any UTXO.

4.8.3 Suggestions

Using a constant for min ADA that will confidently wrap any min ADA for UTXO. However, this isn't flawless and could pose issues in case of related protocol parameter changes.

4.8.4 Implemented solution

Not fixed.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

4.9 Cap the number of script inputs in the fund action

4.9.1 Problem

There is this TODO in the codebase.

That check is useful as it helps prevent some double satisfaction cases. Still, a more straightforward solution would be to cap script inputs in the validator (e.g., only treasury inputs allowed).

4.9.2 Reason for the problem

The current code effectively prevents treasury inputs, but does it in a different way than the rest of the codebase.

4.9.3 Suggestions

The ideal solution would be to replace this check with the cap on script inputs.

4.9.4 Implemented solution

Not fixed.

However, the current approach is functionally equivalent to the proposed solution.

4.10 Constrain redeemers used during fund and reorganize action

4.10.1 Problem

There are this and this TODO comments in the codebase.

We think it would be useful to express the intent directly in the on-chain validator.

4.10.2 Reason for the problem

Preventing potential future double satisfaction and encoding the intent directly on-chain.

4.10.3 Suggestions

Enforce that the redeemers are all the same for inputs coming from the treasury during `fund` and `reorganize`.

4.10.4 Implemented solution

Not fixed.

4.11 Modify does not enforce enough constraints

4.11.1 Problem

The `modify` redeemer is too lax in the checks it enforces. This allows for flexibility, but puts a lot more responsibility on the signers.

4.11.2 Reason for the problem

Currently, the `modify` redeemer is intentionally underconstrained, this is done intentionally to support a variety of use-cases which the vendor and committee can agree to, such as:

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

- funding a new project from unused funds
- vendor returning funds to the treasury
- donation to treasury
- vendor withdrawing payouts

This flexibility, however, comes with an increased burden on the transaction signers (both the vendor and the committee) to ensure that nothing unexpected is happening. While it is always expected that signers check carefully a transaction they are signing, having more on-chain checks reduces the number of things a signer has to look out for, making the process easier for them.

4.11.3 Suggestion

All the mentioned extra actions have their own redeemer, allowing them to happen during `modify` only adds the benefit of having to sign a single transaction, rather than two separate ones. We believe it would make the protocol easier to reason about if `modify` were limited to the single responsibility of modifying unmatured or paused payouts.

4.11.4 Implemented solution

Not fixed.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

5 Audit Summary & Conclusion

Over the course of this audit the MLabs team found 15 vulnerabilities.

Of these:

- 1 is classified as `critical`
- 7 are classified as `high`
- 6 are classified as `medium`
- 1 is classified as `low`

Also, 11 issues were identified that, while not leading directly to an exploitable vulnerability, could cause issues in the future.

It is important to note that the vulnerabilities and issues listed in this report are not exhaustive, as they represent only those identified within the limited time frame allocated for the audit. MLabs has compiled and summarized these findings in the provided report, detailing potential risks and offering recommendations for remediation. While the report is issued without any guarantee, it is our hope that it will serve as a valuable resource for enhancing the security and reliability of the protocol.

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

6 Appendix

6.1 On-chain parameter review

MLabs has reviewed the on-chain parameters that were provided for the 2025 Budget instance of the Treasury Contract. The desired configuration of these parameters is specified in budget-management. MLabs has verified that the build-script for the contract, which instantiates both Treasury and Vendor contracts with the parameters is conformant to the desired configuration (note that both links point to the commit at which this verification has been carried out).

The following steps were taken to verify the configuration to be correct. Once a new instance of the Treasury Contract is deployed, this verification should be done again against the updated parameters.

We verified that:

- The registry token was minted with the correct token name and policy id, and that it was locked at an always False validator.
- Both Vendor and Treasury configurations correctly reference the policy id of the registry token.
- The rest of the Vendor and Treasury configurations are instantiated properly:
  - We verified the permissions for each actions match the ones described by Intersect.
  - We verified the expiration and payout_upperbound dates are set as described by Intersect
- The build script correctly applies the parameters to the scripts

With the current configuration we have certified that building the scripts produces the following hashes:

```
Registry Policy ID    = 9e65e4ed7d6fd86fc4827d2b45da6d2c601fb920e8bfd794b8ecc619
Treasury Contract     = 8583857e4a12ffe1e6f641a1785a0f2f036c565cfbe6ff9db8e5a469
Vendor Contract       = 882cbb37779b7f1f3dd2b7643662de1bfc180baf225ef820c3d8feae
```

6.2 Vulnerability types

The following list gives some more details for each of the vulnerability types reported in this Audit. Keep in mind this list is not exhaustive of all the possible vulnerability types present on Cardano.

6.2.1 Insufficient staking key control

6.2.1.1 Identifier

insufficient-staking-control

6.2.1.2 Property

All scripts explicitly account for staking credentials.

6.2.1.3 Test

A transaction successfully changes or incorrectly sets the staking credential of a UTxO locked by a validator of the protocol. Alternatively, a transaction sets an arbitrary staking credential for an output being locked by an external credential and holding value consumed from the protocol.

6.2.1.4 Potential impact

- Unpredictable addresses
- Illegitimate staking rewards

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

6.2.2 Unbounded datum

6.2.2.1 Identifier

unbounded-datum

6.2.2.2 Property

A transaction can successfully lock in the protocol a legit UTxO with a datum such that its consumption in a second transaction fails due to reaching the network resources constraints.

6.2.2.3 Test

A transaction can successfully lock in the protocol a legit UTxO with a datum such that its consumption in a second transaction fails due to reaching the network resources constraints.

6.2.2.4 Potential impact

- Unspendable outputs
- Protocol halting

6.2.3 Multiple satisfaction

6.2.3.1 Identifier

multiple-satisfaction

6.2.3.2 Property

All scripts consider the totality of inputs to the transaction, as well as the totality of minted value and value withdrawn from staking validators when allowing spending, minting or withdrawing value.

6.2.3.3 Test

A transaction consumes multiple UTxOs, successfully spending the value attributed to each individual UTxO and respecting the conditions under which value could be spent for each individual UTxO, but without respecting the intended aggregate conditions under which the totality of the value could be spent.

More general variations of this test include the cases where the extra value is not being consumed in inputs to the transaction (and therefore subject to validator scripts rules) but rather from minted value controlled by minting polices or value withdrawn from staking validators.

6.2.3.4 Potential impact

- Leaking protocol tokens
- Unauthorised protocol actions

6.2.4 UTxO contention

6.2.4.1 Identifier

utxo-contention

Docusign Envelope ID: 4F986DE7-36C2-4486-85EC-529642F55D42

6.2.4.2 Property

The protocol is designed in such a way that disincentivises the attempt to consume the same UTxO by multiple actors.

6.2.4.3 Test

One out of two or more transactions trying to consume the same UTxO fails due to the UTxO not existing anymore.

6.2.4.4 Potential impact

- Protocol stalling

6.2.5 Deviation From Spec

6.2.5.1 Identifier

deviation-from-spec

6.2.5.2 Property

The implementation matches the specification.

6.2.5.3 Potential impact:

- Incorrect behaviour

6.2.6 Min Ada

6.2.6.1 Identifier

min-ada

6.2.6.2 Property

Calculations involving UTxO values should always account for the minAda that MUST be present in these UTxOs

6.2.6.3 Test

There is some on-chain calculation involving UTxO values that does not correctly account for minAda present in the UTxO.

6.2.6.4 Potential impact

- Protocol stalling
- Protocol halting

Giovanni Garufi

Audit team lead

Signed by:
Giovanni Garufi
68209CE6D9924D8...

7/16/2025